SimiDTR: Deep Trajectory Recovery with Enhanced Trajectory Similarity

No Author Given

No Institute Given

Abstract. The pervasiveness of GPS-equipped smart devices and the accompanying deployment of sensing technologies generates increasingly massive amounts of trajectory data that cover applications such as personalized location recommendation and urban transportation planning. Trajectory recovery is of utmost importance for incomplete trajectories (resulted from the constraints of devices and environment) to enable their completeness and reliability. To achieve effective trajectory recovery, we propose a novel trajectory recovery framework, namely **D**eep **T**rajectory **R**ecovery with enhanced trajectory **Simi**larity (SimiDTR), which is capable of contending with the complex mobility regularity found in trajectories in continuous space. In particular, we design a rule-based information extractor to extract the spatial information related to an incomplete trajectory, which is then fed into a deep model based on attention mechanism to generate a tailored similar trajectory for the incomplete trajectory. Finally, we use a deep neural network model to recover the incomplete trajectory with the blessing of its similar trajectory. An extensive empirical study with real data offers evidence that the framework is able to advance the state of the art in terms of effectiveness for trajectory recovery, especially in scenes with sparse trajectory data.

Keywords: Trajectory Recovery · Trajectory Similarity · Sparse Data.

1 Introduction

The widespread use of various mobile devices has resulted in a proliferation of trajectory data, which contain a wealth of mobility information that is critical to location-based services, e.g., route optimization [30], travel time estimation [7], and POI recommendation [5].

Trajectory data use discrete spatial-temporal point pairs to describe the motion of objects in continuous time and space. Due to the limitations of equipment and environment such as equipment failure and signal missing, many trajectories are recorded at a low sampling rate or with missing locations, called *incomplete trajectories*. Too large sampling interval between two consecutive sampling points can lose detailed information and lead to high uncertainty [31], which affects downstream applications (e.g., indexing [13], clustering [21], and mining [12, 25, 26]) negatively. Therefore, it is important to recover missing spatial-temporal points for incomplete trajectories and reduce their uncertainty.

In general, previous studies on trajectory recovery can be divided into two directions. The first direction focuses primarily on modeling users' transition pattern among different locations to predict users' missing locations [4, 8, 9, 11, 16, 20, 26–28]. The basic task is essentially a classification task, and the recovered trajectories are usually composed of locations or POIs. The second direction aims to recover the specific geographic coordinates of trajectories at the missing timestamps based on the incomplete trajectory data recorded [2, 6, 10, 15, 17–19, 24, 25, 29]. The final rebuilt trajectories usually consist of precise (GPS or road network) coordinates. In this work, we focus on the second direction, i.e., recovering precise GPS coordinates for incomplete trajectories.

A straightforward approach for the second direction is to regard a single trajectory as two-dimensional time series directly and apply time series imputation methods to recover incomplete trajectories [2, 6, 10, 17, 18, 29]. These methods exhaust all the precise information of a single incomplete trajectory when recovering it and work quite well when the proportion of the missing trajectory data is small. However, their effectiveness decreases significantly as the missing proportion increases, which means that they cannot deal with the sparse trajectory data. Another common solution for this problem is cell-based methods [15, 19, 24, 25], which divide the space into discrete cells and then recover the missing trajectories described by cells. They further design different post-calibration algorithms to refine the results. These methods transform the trajectory recovery problem from infinitely continuous space into finite discrete space, which reduce the complexity of prediction to improve the capacity of modeling transition patterns. Although the cell-based methods can alleviate the problem of data sparsity to some extent, they only use the information contained in the incomplete trajectory instead of making full use of information coming from other trajectories. Besides, some extra noise and inaccurate information would inevitably be introduced since these methods use cells to represent trajectories. Furthermore, in the calibration stage, there is a lack of available information for getting accurate trajectory coordinates.

Exploiting the similarity among different trajectories to model the complex mobility regularity for incomplete trajectories, we propose a novel trajectory recovery framework, namely **D**eep **T**rajectory **R**ecovery with enhanced trajectory **Simi**larity (SimiDTR) to recover precise coordinates for trajectories. To address the problem of data sparsity, we carefully design a rule-based information extractor to tease out a raw similar trajectory, which has relevant spatial information about the given incomplete trajectory. This raw similar trajectory is the result of integrating information from several other related incomplete trajectories. Considering the properties (e.g., spatial bias, temporal bias, and temporal shifts) of trajectory data, we use an attention-based deep neural network model to sort out this raw similar trajectory and generate a similar trajectory tailored to the incomplete trajectory (i.e., the similar trajectory that does not actually exist but best fits the data of incomplete trajectory), which is used for recovering the incomplete trajectory. In order to make full use of the trajectory coordinate information, we perform the trajectory recovery in continuous space.

Our contributions can be summarized as followed:

- We propose a novel deep trajectory recovery framework with enhanced trajectory similarity, which directly recovers coordinates of trajectories in continuous space. To our knowledge, this is the first study that takes similar trajectory information into the deep learning model in the field of trajectory recovery.
- To contend with the sparsity of incomplete trajectories, we design a rule-based information extractor that aims to generate a raw similar trajectory for the target incomplete trajectory and propose an attention-based deep neural network model to create a tailor-made similar trajectory based on the inherent characteristics of trajectory data for incomplete trajectory recovery.
- We report on experiments using real trajectory data, showing that our proposal significantly outperforms state-of-the-art baselines in terms of effectiveness for trajectory recovery, especially in scenes with sparse trajectory data.

The remainder of the paper is organized as follows. Section 2 surveys related work, and Section 3 covers preliminaries. The proposed framework is detailed in Section 4, followed by a coverage of experimental results in Section 5. Section 6 concludes the paper.

2 Related Work

In this section, we review the prior studies of trajectory recovery. Based on the objects to be recovered, the trajectory recovery can be classified into location recovery and coordinate recovery.

2.1 Location Recovery

Location recovery aims to predict missing locations (e.g., POIs) for a trajectory [20,26,27], which is of great importance for some downstream location-based applications such as next POI recommendation and route planning. Xi et al. [26] propose a Bi-directional Spatial and Temporal Dependence and users' Dynamic Preferences (Bi-STDDP) framework for location recovery, which takes into account bidirectional spatio-temporal dependency and incorporates POI popularity and dynamic user preferences into the embedding layer. AttnMove [27] firstly uses multiple sparse historical trajectories of a user to aggregate into a dense historical trajectory, then injects the information contained in this dense trajectory into trajectory recovery process using several different attention mechanisms. On the basis of AttnMove, PeriodicMove [20] takes into account the influence of trajectory shifting periodicity and further uses Gated GNN to capture transition patterns among locations.

2.2 Coordinate Recovery

Taking the trajectory data as two-dimensional time series, the time series imputation methods [2,6,10,17,18,29] can be used to recover the missing coordinates (observations) in trajectories (time series). Two typical RNN-based methods, M-RNN [29] and BRITS [2], use the hidden states of bidirectional RNN to impute missing values of time series. The former regards missing values as constants, while the latter imputes series values by regarding them as variables of the RNN calculating graph. SAITS [6] is currently a state-of-the-art method in this field, which is a self-attention-based model and uses a joint-optimization training method of imputation and reconstruction to impute missing values. There are also some GAN-based methods [17,18] and VAE-based methods [10], which can be used to recover coordinates for trajectories. However, they fail to solve the data sparsity problem. Their performance will deteriorate when recovering sparse incomplete trajectory data that have massive missing coordinates.

Generating the recovered trajectories represented by cells, following by a post-calibration algorithm to get the coordinates of trajectories, Cell-based methods [15,19,24,25] are proposed for this task. For example, Wei et al. [25] and Liu et al. [15] build a top-k routes inference framework and an answering system by aggregating uncertain trajectories, respectively. After obtaining the trajectory represented in the cell level, they find the trajectories passing through the sequence of cells sequentially, fit the line segments by linear regression, and stitch the segments together. Wang et al. [24] and Ren et al. [19] propose a deep learning model, which utilizes a traditional seq2seq framework and attention mechanism. The former uses Kalman filtering as the post-calibration module. The later inputs cell-level trajectories into a deep learning model and predicts the road segment ID and moving ratio directly in the road network. Inevitably, however, the cell-based methods would introduce extra noise and inaccurate information. During the post-calibration stage, there is a lack of available information for getting the accurate coordinates of missing trajectory points. Moreover, they only focus on the spatio-temporal information contained in the incomplete trajectory while ignoring the useful information of the similar trajectories.

In this study, we will focus on the GPS coordinate recovery for trajectories. To better ensure the coordinate recovery quality, it is attractive to integrate the spatial information of similar trajectories into the incomplete trajectory recovery, so that the missing coordinates can be estimated accurately. Complementing existing studies, our study aims to utilize the similarity between the target incomplete trajectory and its similar trajectories to enable more accurate coordinate recovery.

3 Problem Statement

In this section, we present necessary preliminaries and then define the problem addressed.

Definition 1 (Location) *A location is denoted by $l = (lon, lat)$, where lon and lat represent its longitude and latitude, respectively.*

Definition 2 (Region) *We divide the geographical space into a set of discrete and disjoint square regions, denoted by R. Each region (also referred to as grid or cell), denoted by $r \in R$, is a small square with the length of Δr.*

Definition 3 (Trajectory Point) *A trajectory point is a sampled point from a moving object, denoted by $p = (lon, lat, t)$, where (lon, lat) is the location of this point, and t denotes its sampling timestamp.*

If the location of a trajectory point is known, it is *a recorded trajectory point* (*a recorded point* for short), and we can get the region r where the point falls. Otherwise, it is *a missing trajectory point* (*a missing point* for short), denoted by $\tilde{p}$, where the sampling timestamp is given but the location (i.e., the longitude and latitude) is unknown.

Definition 4 (Sampling Interval) *The sampling interval, denoted by ϵ, is the time difference between two consecutive trajectory points, which is generally set according to the sampling equipment.*

Ideally, the sampling interval of trajectory data is a fixed constant, but due to the inherent temporal bias of trajectory data, the sampling interval often changes near ϵ in most cases, i.e., the sampling interval often slightly larger or smaller than ϵ.

Definition 5 (Complete Trajectory) *A complete trajectory, denoted by $tr = p_1 \rightarrow p_2 \rightarrow ... \rightarrow p_i \rightarrow ... \rightarrow p_n$, is a sequence of trajectory points sampled from a moving object, where p_i is the i-th trajectory point of tr.*

Under ideal condition, given a fixed time interval ϵ, we can get $\epsilon = p_{i+1}.t - p_i.t$ $(0 \leq i \leq n-1)$, where $p_i.t$ denotes the timestamp of p_i.

Definition 6 (Incomplete Trajectory) *An incomplete trajectory is composed of a sequence of recorded trajectory points and missing trajectory points, denoted by $\tilde{tr} = p_1 \rightarrow \tilde{p_2} \rightarrow p_3 \rightarrow \tilde{p_4} \rightarrow p_5 \rightarrow \cdots$, where $p_1, p_3, p_5 \cdots$ and $\tilde{p_2}, \tilde{p_4} \cdots$ are the recorded and the missing points of the incomplete trajectory, separately.*

Problem Statement. Given a set of incomplete trajectories $\tilde{Tr}$ with sampling interval ϵ, our problem is to recover their missing coordinates.

4 Methodology

We propose a framework, namely **D**eep **T**rajectory **R**ecovery with enhanced trajectory **Simi**larity (SimiDTR), to recover coordinates for incomplete trajectories. We first give an overview of the framework and then provide specifics on each component in the framework.

4.1 Framework Overview

The SimiDTR framework consists of four components, as shown in Fig. 1.

Rule-based Information Extractor. Given an incomplete trajectory and to inject more relevant spatial information into the subsequent components, we design a rule-based information extractor that fills in missing trajectory points of the incomplete trajectory in continuous space so that generates a related trajectory (called raw similar trajectory), which is the result of integrating information from several other related incomplete trajectories. This raw similar trajectory is simply a rigid combination of relevant spatial information.

Trajectory Embedding Layer. In order to encode the sequence of trajectory coordinates (including raw similar and incomplete trajectory coordinates) into high-dimensional embeddings and introduce sequential, temporal and spatial information simultaneously, we design a trajectory embedding layer, which is composed of three kinds of embeddings.

Encoder. Consider that trajectory data exhibit inherent spatial bias, temporal bias, and temporal shifts even when they are collected under the same sampling interval [14], which means that the locations and timestamps of the corresponding trajectory points in two trajectories (even if capturing the same motion of an object) are usually not the same. To solve this issue, we feed the raw similar trajectory into an attention-based encoder to make it sorted out. The encoder takes the embeddings of the raw similar trajectory and incomplete trajectory as input and outputs a similar trajectory represented by high-dimensional vectors, which is tailored for the incomplete trajectory.

Decoder. In the recovery step, we send the high-dimensional vectors of the similar trajectory and the embedding of the incomplete trajectory to a decoder and finally use a linear layer to estimate the missing coordinates for getting the recovered trajectory.

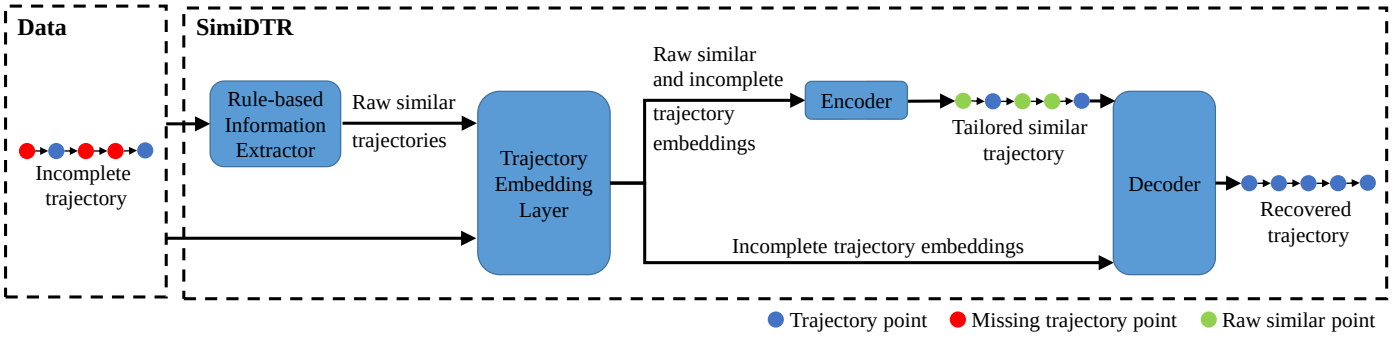


Fig. 1. The framework of SimiDTR model.

4.2 Rule-base Information Extractor

The rule-base information extractor extracts relevant spatial information from the entire dataset for the input incomplete trajectory, so as to generate its raw similar trajectory. Its framework is shown in Fig. 2, and its pseudocode is shown in Algo.1.

We firstly divide the input trajectory into several segments. These trajectory segments have the same form that both sides are recorded trajectory points and the middle is missing trajectory points. For example, incomplete trajectory $\tilde{tr} = \tilde{p_1} \rightarrow p_2 \rightarrow \tilde{p_3} \rightarrow \tilde{p_4} \rightarrow p_5 \rightarrow p_6 \rightarrow p_7 \rightarrow \tilde{p_8} \rightarrow \tilde{p_9}$ is divided into segments $\tilde{p_1} \rightarrow p_2$, $p_2 \rightarrow \tilde{p_3} \rightarrow \tilde{p_4} \rightarrow p_5 \rightarrow p_6 \rightarrow p_7$ and $p_5 \rightarrow p_6 \rightarrow p_7 \rightarrow \tilde{p_8} \rightarrow \tilde{p_9}$. Then the divided trajectory segments is sent to *Padding Module* separately. The padding module finds other relevant trajectory segments from dataset to fill in all or part of the missing points according to the information that the selected relevant trajectory segment contains. Afterwards, we splice the outputs of padding module together and get the raw similar trajectory which is filled once. Since padding module only extracts the information from the most relevant trajectory segment in a single segment padding process, part of the missing points may not be filled because there is not enough information in the only one segment in the sparse scenario. So we regard the filled trajectory points as recorded points and continue to input the raw similar trajectory into padding module until all the missing points are filled or the outputs no longer changes.

The output that no longer changes means that there is no relevant information available in the trajectory dataset to fill in the missing points. In this case, we use the module *Linear Interpolation* to fill in the missing trajectory points.

In the following, we give more details about the padding module and the linear interpolation.

Algorithm 1: Rule-based Information Extractor

Input: Incomplete trajectory
Output: Raw similar trajectory
do
  Divide incomplete trajectory into segments;
  Fill in segments with padding module separately;
  Splice segments into trajectory;
while *Input and output of the loop are different*;
if *Unfilled points exist* **then**
  Fill in the missing points with linear interpolation;
return Filled trajectory (i.e. raw similar trajectory)

Padding Module The padding module takes a trajectory segment as input, and fill in all or part of its missing points at a time. We call this input trajectory

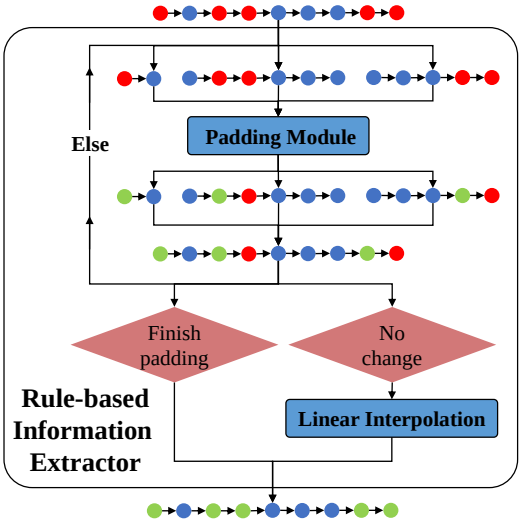


Fig. 2. The framework of Information Extractor.

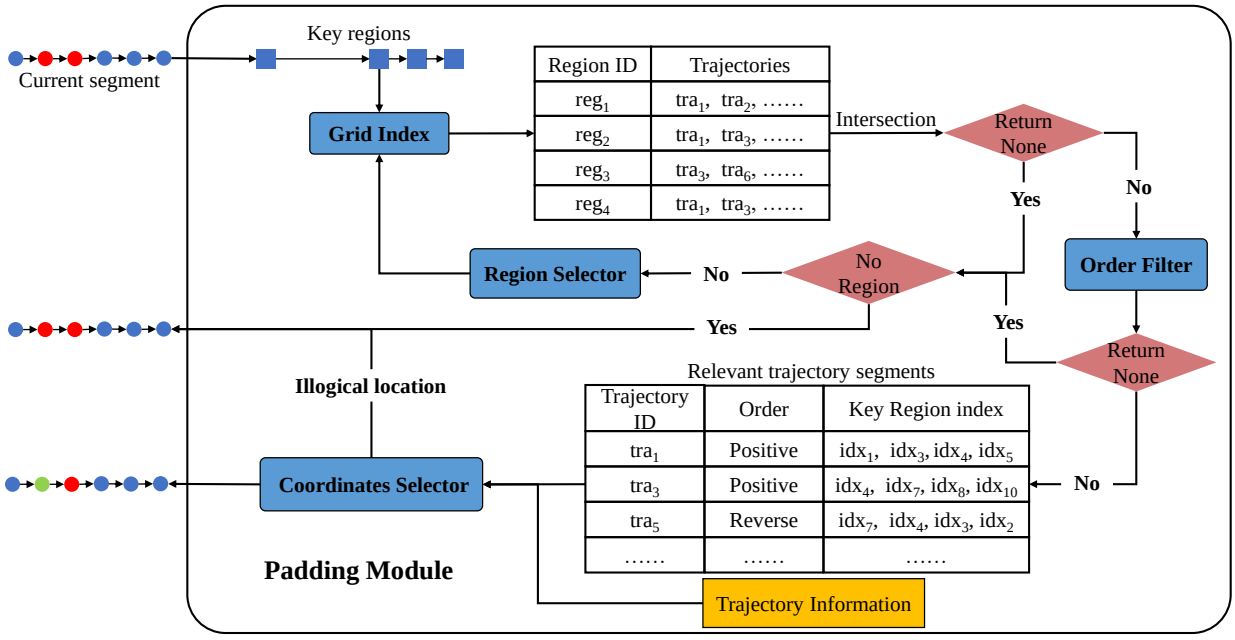


Fig. 3. The process of Padding Module.

segment current segment, and call its corresponding trajectory current trajectory. As shown in Fig. 3, we firstly change current segment into region-level. We call these recorded points key points and call their corresponding regions key regions since they are the main basis for selecting relevant space information. With the help of *Grid Index*, we can quickly get the IDs of trajectories (except current trajectory) through key regions and the indexes of these regions in these trajectories. With these indexes, we are able to retrieve the trajectory segments through the key regions anytime. Relying on *Order Filter*, we can remove trajectory segments with no available space information and only retain the trajectory segment through key regions in the positive order and reverse order. These are the trajectory segments contain the space information we can utilize, which are called relevant trajectory segments. Taking relevant trajectory segments and the trajectory information as input, *Coordinates Selector* can get the proper longitude-latitude pairs to fill in the missing trajectory points. Here, the trajectory information is composed of the sequence of coordinates, regions, timestamps and indexes of all the incomplete trajectories. If there is no trajectory segment that we can utilize, we should reduce the number of key points and regions. *Region Selector* can help us remove one point/region, which allows us to continue above process.

Grid Index To facilitate retrieval, we build a grid index shown in Fig. 4. We first divide the entire geographic space into disjoint regions. For each region we record the IDs of trajectories that whose recorded trajectory points fall inside it. And for each trajectory, we record the indexes of the region, and sort them in the order that the region appears in the trajectory. Several indexes in a trajectory means the trajectory visits the region several times. Here, the index of regions is the same to the index of trajectory points. For example, a trajectory denoted by $tr = p_1 \rightarrow p_2$ consists of two trajectory points. It can also be denoted by $tr = r_1 \rightarrow r_2$ in the region-level. Here, idx_1 represents the order of p_1 and r_1 in tr.

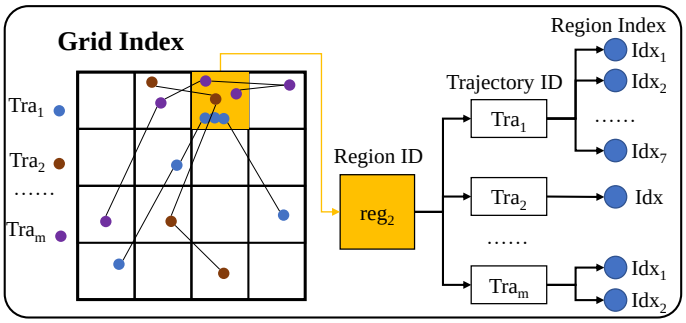


Fig. 4. The Schematic Diagram of Grid Index.

Order Filter The purpose of order filter is to only retain the trajectory segments through the key regions in the positive and reverse order, and to remove the trajectory segments that have no available information. The outputs of order filter are relevant trajectory segments. For example, current segment is $p_1 \rightarrow p_2 \rightarrow \tilde{p_3} \rightarrow \tilde{p_4} \rightarrow p_5 \rightarrow p_6$, and the key regions are $r_1 \rightarrow r_2 \rightarrow r_5 \rightarrow r_6$. In the case, we only retain the trajectory segments who pass through the key regions in the order of $r_1 \rightarrow r_2 \rightarrow r_5 \rightarrow r_6$ and $r_6 \rightarrow r_5 \rightarrow r_2 \rightarrow r_1$. Suppose there is a trajectory segment $p'_1 \rightarrow p'_2 \rightarrow p'_5 \rightarrow p'_6$ and these four points fall in the four key regions separately. If there is no recorded trajectory points between p'_2 and p'_5, we will remove it.

To achieve this purpose, we firstly use grid index to get the trajectories through key regions. For each such trajectory, we can also get the indexes of key regions in this trajectory. Each region usually corresponds to several indexes, which means this trajectory passes through the region several times. For each region of these trajectories, we keep only one index and eventually form the relevant trajectory segments in the same or opposite order as the key regions. When searching for the positive relevant trajectory segment, we regard the smallest index of the first region as the index of the first trajectory point in relevant trajectory segment. In each subsequent step, we use the smallest but larger than the previous index of the region as the selected index. Searching for the reverse relevant trajectory segment is the opposite of the process. We regard the largest index of the first region as the first index. In the next steps, we select the largest but smaller than previous indexes. Lastly, we remove segments without available trajectory points.

For example, as shown in Fig. 5, current segment is $p_1 \rightarrow p_2 \rightarrow \tilde{p_3} \rightarrow \tilde{p_4} \rightarrow p_5 \rightarrow p_6$, and the key regions are $r_1 \rightarrow r_2 \rightarrow r_5 \rightarrow r_6$. Suppose trajectory tr_1 passes through the four key regions, with the help of grid index, we get its indexes of key regions. There are two indexes of r_2, which mean the 5th and 13th trajectory points of tr_1 visit r_2. As we searching for the positive relevant trajectory segment, we first select idx_2 for r_1. In the next steps, we select idx_5, idx_7, idx_9 for r_2, r_5, r_6 separately. In the same way, we can get the reserve index sequence $idx_{16}, idx_{13}, idx_{12}, idx_9$. Suppose the 6th trajectory point of tr_1 is a recorded trajectory point, then we get a positive relevant trajectory point generated by tr_1. As there is no recorded trajectory point between the 13-th and 12-th trajectory points of tr_1, we filter the reverse relevant trajectory segment out.

Coordinates Selector Coordinates selector takes the relevant trajectory segments and trajectory information (i.e., the GPS coordinates, timestamps, regions of all the trajectories) as input, output one raw similar trajectory segment at a time.

Process. Here, the recorded trajectory points in relevant trajectory segment who fall in key regions are called relevant points. We firstly calculate the average distance between key points and relevant points. We think the smaller this value, the higher the correlation between them. So as shown in Fig. 6, we sort relevant trajectory segments in order of this average distance from smallest to largest. Then we select the GPS coordinate values from the most relevant trajectory

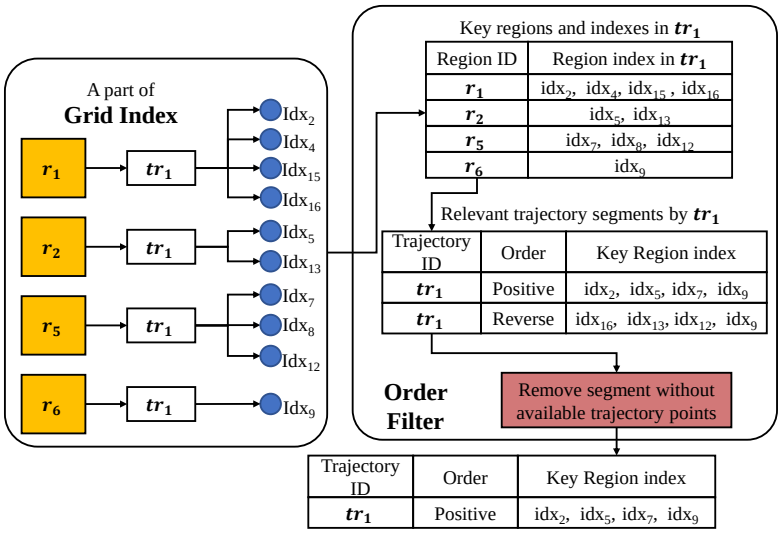


Fig. 5. The Schematic Diagram of Order Filter.

segment to fill in the missing points. We design coordinate point filling strategies separately in the following three scenarios.

- Scenario 1: Both sides of the current segment are recorded points, with missing points in the middle.
- Scenario 2: The current segment is preceded by recorded points and followed by missing points. This occurs when the trajectory points at the end of the trajectory are missing.
- Scenario 3: The current segment is preceded by missing points and followed by recorded points. This occurs when the trajectory points at the beginning of the trajectory are missing.

Finally, we judge whether these filled points are illogical. If the velocity of a moving object between any two trajectory points exceeds a threshold, we consider it illogical. Then we remove the filled missing points and use the left relevant trajectory segments to repeat the above process. In Scenario 1 we set this speed threshold to 2km/min, and in other two scenarios we set it to 1.5km/min. If there is no relevant trajectory segment providing logical points, we will not fill any of the missing points.

Missing Points Filling. We mainly choose the appropriate GPS coordinates for the missing points based on time ratio, denoted by $time\_ratio$.

In Scenario 1, as shown in Fig. 7, we assume the indexes of the two key points which are adjacent to missing points are bc and ec. The indexes of their corresponding relevant points of positive segment and reverse segment are bp, ep and br, er separately. Then the time ratio of missing point is $time\_ratio_c^i = \frac{t_i - t_{bc}}{t_{ec} - t_{bc}}$, $bc < i < ec$. Here, i means the index of a missing point in incomplete

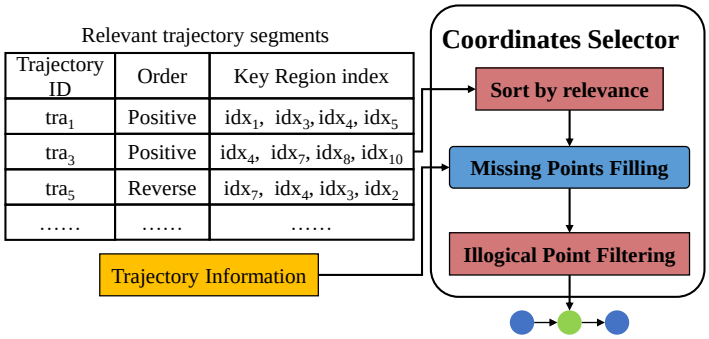


Fig. 6. The Schematic Diagram of Coordinates Selector.

trajectory. The time ratio of available points in positive segment and reverse segment are $time\_ratio_p^j = \frac{t_j - t_{bp}}{t_{ep} - t_{bp}}$, $bp < j < ep$ and $time\_ratio_r^k = \frac{t_{br} - t_k}{t_{br} - t_{er}}$, $er < k < br$. We select the closest $time\_ratio_p^j$ or $time\_ratio_r^k$ for each $time\_ratio_c^i$ and fill in the missing $\tilde{p}_i$ with the recorded point corresponding to $time\_ratio_p^j$ or $time\_ratio_r^k$. We fill in all or part of missing points according to the relevant segment we select.

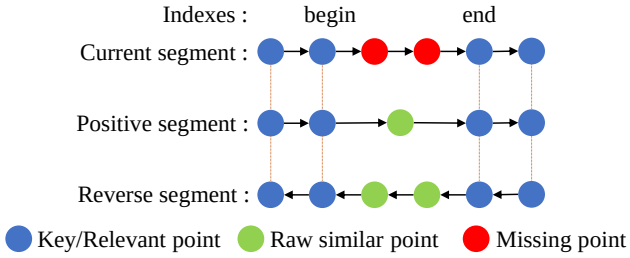


Fig. 7. Scenario 1: Fill in the middle missing points.

In Scenario 2, as shown in Fig. 8, we only fill in the missing point which is adjacent to key points. If there are at least two key points, we assume the indexes of the two key points which are adjacent to missing points are $bc1$ and $bc2$. The indexes of their corresponding relevant points of positive segment and reverse segment are $bp1$, $bp2$ and $br2$, $br2$ separately. Then the time ratio of the first missing point is $time\_ratio_c^i = \frac{t_i - t_{bc2}}{t_{bc2} - t_{bc1}}$, $i = bc2 + 1$. The time ratio of available points in positive segment and reverse segment are $time\_ratio_p^j = \frac{t_j - t_{bp2}}{t_{bp2} - t_{bp1}}$, $bp2 < j$ and $time\_ratio_r^k = \frac{t_{br2} - t_k}{t_{br1} - t_{br2}}$, $k < br2$. Then we can find the closest $time\_ratio_p^j$ or $time\_ratio_r^k$ for $time\_ratio_c^i$ and fill in the missing point. If there is only one key point, the three time ratios are $time\_ratio_c^i =$

$t_i - t_{bc2},\ i = bc2 + 1,\ time\_ratio_p^j = t_j - t_{bp2},\ bp2 < j$ and $time\_ratio_r^k = t_{br2} - t_k,\ k < br2$ separately.

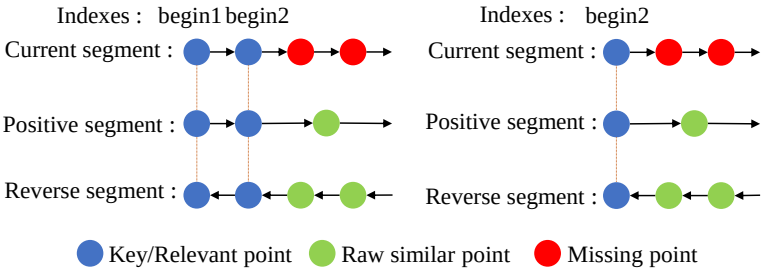


Fig. 8. Scenario 2: Fill in the next missing point.

In Scenario 3, as shown in Fig. 9, we also only fill in the missing point which is adjacent to key points. If there are at least two key points, we assume the indexes of the two key points which are adjacent to missing points are $ec1$ and $ec2$. The indexes of their corresponding relevant points of positive segment and reverse segment are $ep1$, $ep2$ and $er2$, $er2$ separately. Then the time ratio of the last missing point is $time\_ratio_c^i = \frac{t_{ec1} - t_i}{t_{ec2} - t_{ec1}},\ i = ec1 - 1$. The time ratio of available points in positive segment and reverse segment are $time\_ratio_p^j = \frac{t_{ep1} - t_j}{t_{ep2} - t_{ep1}},\ j < ep1$ and $time\_ratio_r^k = \frac{t_k - t_{er1}}{t_{er1} - t_{er2}},\ er1 < k$. Then we can find the closest $time\_ratio_p^j$ or $time\_ratio_r^k$ for $time\_ratio_c^i$ and fill in the missing point. If there is only one key point, the three time ratios are $time\_ratio_c^i = t_{ec1} - t_i,\ i = ec1 - 1,\ time\_ratio_p^j = t_{ep1} - t_j,\ j < ep1$ and $time\_ratio_r^k = t_k - t_{er1},\ er1 < k$ separately.

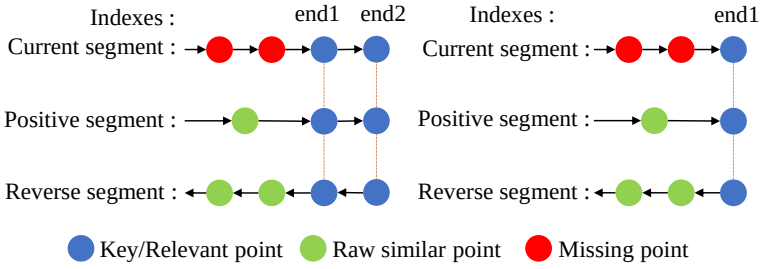


Fig. 9. Scenario 3: Fill in the last missing point.

Region Selector In the following three cases, we should use region selector to remove one key point, because there is no trajectory segment in dataset that is so similar to current segment.

- Case 1: With the exception of current segment, no other trajectory segment passes through all the key regions.
- Case 2: The Case 1 is satisfied, but none of these trajectory segments with available information pass through the key regions in positive or reverse order.
- Case 3: The Case 2 is satisfied, but there are some illogical trajectory points in the selected trajectory segments.

We believe that the closer the recorded point to the missing points, the more critical it is. So, each time we will remove the one point furthest from the missing points. For example, the current segment is $p_1 \rightarrow p_2 \rightarrow \tilde{p_3} \rightarrow \tilde{p_4} \rightarrow p_5$, and it satisfies one of the cases above when p_1, p_2, p_5 are key points. Now, we remove trajectory point p_1 and make p_2, p_5 the key points to continue to fill in the missing points $\tilde{p_3}, \tilde{p_4}$.

Linear Interpolation If there is no relevant information available in the trajectory dataset to fill in the missing points, it's time to use linear interpolation to fill them. For missing points sandwiched in the middle by recorded points, we fill in with one-dimensional linear interpolation of longitude and latitude, respectively. If there are missing points at the beginning and end of the incomplete trajectory, we use the closest recorded points of the incomplete trajectory to fill in them.

4.3 Trajectory Embedding Layer

The responsibility of the trajectory embedding layer is to embed trajectory into dense representations as the input of the following modules. In order to introduce sequence, time and coordinates information simultaneously, we use three kinds of embeddings in total, i.e., positional, timestamp, and coordinate embeddings.

For each trajectory $tr = p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_n,\ n \in N$, where n is the length of tr, N is length of the longest trajectory in dataset. The locations and timestamps of tr are $\boldsymbol{L} \in \mathbb{R}^{n \times 2}$ and $\boldsymbol{T} \in \mathbb{R}^{n \times 1}$ separately.

Positional Embedding The positional embedding is composed of sine and cosine functions following [22], which contains the sequence information of trajectories. We use $\boldsymbol{E}_{idx} \in \mathbb{R}^{n \times d}$ to represents the positional embedding of tr, and generate it as follows,

$$\begin{cases} \boldsymbol{E}_{idx}(i, 2j) = sin(\frac{i}{10000^{2j/d}}) \\ \boldsymbol{E}_{idx}(i, 2j+1) = cos(\frac{i}{10000^{2j/d}}) \end{cases} \tag{1}$$

where i denotes the i-th trajectory point of tr, j denotes the j-th dimension. The positional embedding vector of a trajectory point has d dimensions.

Timestamp Embedding The timestamp embedding is mainly to inject time features of trajectories into the following module. For each timestamps t of $\boldsymbol{T}$, we can calculate the minute of the current hour (denoted by $t_{min} \in \boldsymbol{T}_{min} \in \mathbb{R}^{n\times 1}$) and the second of the current minute (denoted by $t_{sec} \in \boldsymbol{T}_{sec} \in \mathbb{R}^{n\times 1}$) for this timestamp. In other word, t is at the t_{min}-th minute of the current hour the t_{sec}-th second of the current minute. In order to get timestamp embedding of tr, we map them to the interval $[-0.5, 0.5]$ and follow a linear transformation, just as follows,

$$\boldsymbol{E}_t = (Concat([\frac{\boldsymbol{T}_{min}}{59} - 0.5, \frac{\boldsymbol{T}_{sec}}{59} - 0.5]))\boldsymbol{W}_t^T + \boldsymbol{b}_t \tag{2}$$

where $Concat(\cdot)$ means concatenate matrices horizontally. And $\boldsymbol{W}_t \in \mathbb{R}^{d\times 2}$ and $\boldsymbol{b}_t \in \mathbb{R}^{1\times d}$ are linear transformation matrix and bias separately.

Coordinate Embedding In order to ensure that the missing trajectory points (padded by $\boldsymbol{0} \in \mathbb{R}^{1\times 2}$) remain $\boldsymbol{0}$ ($\boldsymbol{0} \in \mathbb{R}^{1\times d}$) vectors when they are mapped to high-dimensional space, we use one-dimensional convolution (denoted by $Conv1d(\cdot)$) to act as a mapping function as follows,

$$\boldsymbol{E}_{GPS} = Conv1d(\boldsymbol{L}) \tag{3}$$

where $Conv1d(\cdot)$ has no bias, and the size of its convolution kernel is 1.

Embedding Integration Assume the raw similar trajectory of tr is denoted by tr_{rs}. Because it is simply a rigid combination of relevant spatial information, which means $\boldsymbol{T}$ doesn't fit tr_{rs} in the missing points, we don't introduce the timestamp embedding of tr_{rs} into the following module. In a word, the final embedding of tr_{rs} and tr are as following,

$$\begin{cases} \boldsymbol{X}_{rs} = \boldsymbol{E}_{idx} + Conv1d(\boldsymbol{L}_{rs}) \\ \boldsymbol{X} = \boldsymbol{E}_{idx} + \boldsymbol{E}_t + \boldsymbol{E}_{GPS} \end{cases} \tag{4}$$

where $\boldsymbol{L}_{rs} \in \mathbb{R}^{n\times 2}$ is the locations of tr_{rs}, and $\boldsymbol{X}_{rs}, \boldsymbol{X} \in \mathbb{R}^{n\times d}$.

4.4 Encoder and Decoder

The framework of this module is shown in Fig. 10. Encoder takes the embedding of Incomplete trajectory and its raw similar trajectory (denoted by $\boldsymbol{X}$ and $\boldsymbol{X}_{rs}$ separately) as input, outputs the tailored similar trajectory represented in the form of a high-dimensional vectors, which is denoted by $\boldsymbol{X}_s \in \mathbb{R}^{n\times d}$. Decoder takes $\boldsymbol{X}$ and its tailored similar trajectory $\boldsymbol{X}_s$ as input, and outputs the recovered trajectory. We use a linear transformation to get its GPS coordinates. Next, we firstly introduce the operation of each component, and then introduce the complete process of encoder and decoder. Many underlying components are following [22].

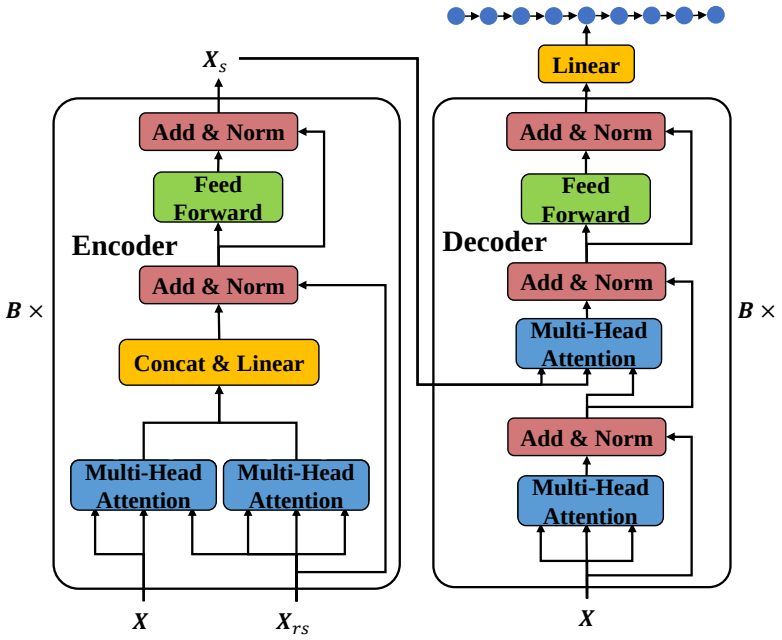


Fig. 10. The Schematic Diagram of Encoder and Decoder.

Multi-Head Attention Assume this module has H heads, its inputs are $\boldsymbol{X}_q, \boldsymbol{X}_k, \boldsymbol{X}_v \in \mathbb{R}^{n\times d}$. For its h-th head, there are operations as following,

$$\begin{aligned}
\boldsymbol{Q}^{(h)} &= \boldsymbol{X}_q \boldsymbol{W}_q^{(h)} \\
\boldsymbol{K}^{(h)} &= \boldsymbol{X}_k \boldsymbol{W}_k^{(h)} \\
\boldsymbol{V}^{(h)} &= \boldsymbol{X}_v \boldsymbol{W}_v^{(h)} \\
\boldsymbol{A}^{(h)} &= SoftMax(\boldsymbol{Q}^{(h)} \boldsymbol{K}^{(h)T}) \\
\boldsymbol{O}^{(h)} &= \boldsymbol{A}^{(h)} \boldsymbol{V}^{(h)}
\end{aligned} \tag{5}$$

where $\boldsymbol{W}_q^{(h)}, \boldsymbol{W}_k^{(h)}, \boldsymbol{W}_q^{(h)} \in \mathbb{R}^{d\times d}$ are the linear transformation matrix to be learned.

Then we concatenate all the $\boldsymbol{O}^{(h)}$ horizontally and use a linear transformation to reshape it, just as follows,

$$\boldsymbol{O}_a = Concat([\boldsymbol{O}^{(1)}, \boldsymbol{O}^{(2)}, \cdots, \boldsymbol{O}^{(H)}])\boldsymbol{W}_a \tag{6}$$

where $Concat(\cdot)$ means concatenate matrices horizontally. $\boldsymbol{W}_a \in \mathbb{R}^{dH\times d}$ is the transformation matrix.

For the sake of narrative, we note all the operations of this module as $\boldsymbol{O}_a = MHA(\boldsymbol{X}_q, \boldsymbol{X}_k, \boldsymbol{X}_v)$.

Concat and Linear This module is mainly to integrate the results of two multi-head attention in encoder. Suppose these two results are $\boldsymbol{O}_{a1}, \boldsymbol{O}_{a2} \in \mathbb{R}^{n \times d}$, the operations of this module is as following,

$$\boldsymbol{O}_c = Concat([\boldsymbol{O}_{a1}, \boldsymbol{O}_{a2}])\boldsymbol{W}_c + \boldsymbol{b}_c \tag{7}$$

where $\boldsymbol{W}_c \in \mathbb{R}^{2d \times d}$ and $\boldsymbol{b}_c \in \mathbb{R}^{1 \times d}$ are linear transformation matrix and bias. All the operations in this module are noted by $\boldsymbol{O}_c = CL(\boldsymbol{O}_{a1}, \boldsymbol{O}_{a2})$

Add and Norm In this module, Add and Norm means residual connection and layer normalization separately. Suppose it follows function $F(\cdot)$, and $\boldsymbol{X}_f \in \mathbb{R}^{n \times d}$ is the input of $F(\cdot)$. The operations of this module are as following,

$$\begin{aligned} \boldsymbol{O}_{Add} &= \boldsymbol{X}_f + Dropout(F(\boldsymbol{X}_f)) \\ \boldsymbol{O}_A &= \frac{\boldsymbol{O}_{Add} - \boldsymbol{\mu}_{Add}}{\boldsymbol{\sigma}_{Add}} \end{aligned} \tag{8}$$

where $\boldsymbol{\mu}_{Add}, \boldsymbol{\sigma}_{Add} \in \mathbb{R}^{n \times 1}$ are the mean and standard deviation of $\boldsymbol{O}_{Add}$. We note the operations of linear normalization as $\boldsymbol{O}_A = LN(\boldsymbol{O}_{Add})$.

Feed Forward Suppose the input of this module is $\boldsymbol{X}_F \in \mathbb{R}^{n \times d}$, the operations are as following,

$$\begin{aligned} \boldsymbol{O}_1 &= \boldsymbol{X}_F \boldsymbol{W}_1 + \boldsymbol{b}_1 \\ \boldsymbol{O}_2 &= Relu(\boldsymbol{O}_1) \\ \boldsymbol{O}_F &= \boldsymbol{O}_2 \boldsymbol{W}_2 + \boldsymbol{b}_2 \end{aligned} \tag{9}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{d \times d'}, \boldsymbol{W}_2 \in \mathbb{R}^{d' \times d}, \boldsymbol{b}_1 \in \mathbb{R}^{1 \times d'}, \boldsymbol{b}_2 \in \mathbb{R}^{1 \times d}$ are transformation matrices and biases. All the operations of this module are noted by $\boldsymbol{O}_F = FFN(\boldsymbol{X}_F)$.

Encoder In encoder, we are equivalent to refer the change in $\boldsymbol{X}$, and use attention mechanism to generate a similar trajectory tailored to it. Encoder is composed of $B \in \mathbb{R}$ encoder blocks. For each block, its process is as following,

$$\begin{aligned} \boldsymbol{O}_{a1} &= MHA(\boldsymbol{X}_{rs}, \boldsymbol{X}, \boldsymbol{X}) \\ \boldsymbol{O}_{a2} &= MHA(\boldsymbol{X}_{rs}, \boldsymbol{X}_{rs}, \boldsymbol{X}_{rs}) \\ \boldsymbol{O}_c &= CL(\boldsymbol{O}_{a1}, \boldsymbol{O}_{a2}) \\ \boldsymbol{O}_A &= LN(\boldsymbol{X}_{rs} + Dropout(\boldsymbol{O}_c)) \\ \boldsymbol{X}_{rs} &= LN(\boldsymbol{O}_A + Dropout(FFN(\boldsymbol{O}_A))) \end{aligned} \tag{10}$$

We note the $\boldsymbol{X}_{rs}$ outputed by the last Encoder Block as $\boldsymbol{X}_s$, which is the tailored similar trajectory of $\boldsymbol{X}$.

Decoder In decoder, We are equivalent to use attention mechanism to adding additional relevant spatial information when recovering $\boldsymbol{X}$. Decoder consists of $B \in \mathbb{R}$ decoder blocks. For each block, its process is as following,

$$\begin{aligned}
\boldsymbol{O}_{a1} &= MHA(\boldsymbol{X}, \boldsymbol{X}, \boldsymbol{X}) \\
\boldsymbol{O}_{A1} &= LN(\boldsymbol{X} + Dropout(\boldsymbol{O}_{a1})) \\
\boldsymbol{O}_{a2} &= MHA(\boldsymbol{O}_{A1}, \boldsymbol{X}_s, \boldsymbol{X}_s) \\
\boldsymbol{O}_{A2} &= LN(\boldsymbol{O}_{A1} + Dropout(\boldsymbol{O}_{a2})) \\
\boldsymbol{X} &= LN(\boldsymbol{O}_{A2} + Dropout(FFN(\boldsymbol{O}_{A2})))
\end{aligned} \tag{11}$$

Finally, we send the $\boldsymbol{X}$ outputed by the last decoder block into a linear layer to get the coordinates of the predicted trajectory:

$$\boldsymbol{L}_{pre} = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b} \tag{12}$$

where $\boldsymbol{W} \in \mathbb{R}^{d \times 2}, \boldsymbol{b} \in \mathbb{R}^{1 \times 2}$ are the transformation matrix and bias.

4.5 Training

We simply use mean square error between the locations of predicted trajectory (denoted by $\boldsymbol{L}_{pre} \in \mathbb{R}^{n \times 2}$) and the locations of true trajectory (denoted by $\boldsymbol{L}_{tar} \in \mathbb{R}^{n \times 2}$) as the loss function:

$$\mathcal{L} = \frac{1}{2n} \sum_{i=1}^{n} \sum_{j=1}^{2} (predict_{i,j} - target_{i,j})^2 \tag{13}$$

where $predict_{i,j}, target_{i,j}$ are the elements of row i and column j of $\boldsymbol{L}_{pre}$ and $\boldsymbol{L}_{tar}$, respectively.

5 Experimental Evaluation

5.1 Experimental Setup

Datasets. To verify the effect of our proposed model, we use two real-world taxi trajectory datasets collected from Porto and Shanghai, respectively. The Porto dataset[1] (called Porto for short) is a public trajectory dataset, which is released for a taxi trajectory prediction competition on Kaggle. The Shanghai dataset[2] (called Shanghai for short) is also a public trajectory dataset collected by the Hong Kong University Of Science and Technology (HKUST), which is composed of several parts, e.g., taxis, buses, and mobile phone data. We only use its taxi data.

For the Porto dataset, we first remove records with empty trajectories. The sampling interval of the original dataset is 15 seconds, and we sample every 4

[1] https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i
[2] https://cse.hkust.edu.hk/scrg/

points to convert the interval to 1 minute. We set the longitude and latitude range from -8.735 to -8.156 and 40.953 to 41.308, respectively. Next, we remove all trajectories that contain points being out of the latitude and the longitude ranges. We set the edge length Δr of each region to 0.001, which is about 111 meters. If the number of trajectory points in the region is less than 2, we remove such regions together with the trajectory points in them. In the end, we only keep trajectories between 10 and 30 in length.

For the Shanghai taxi dataset, we regard the stay points of taxis as boundaries to segment their entire day's trajectories. We set the longitude and latitude range from 121.25 to 121.75 and 30.85 to 31.50, respectively, and also remove all trajectories containing trajectory points that are out of the latitude and the longitude ranges. The edge length Δr of each region is also set to 0.001, and the regions containing less than 15 trajectory points together with the trajectory points are removed. In this dataset, we only keep the trajectories with the number of trajectory points between 10 and 100.

Note that after preprocessing, the time interval of Porto is a constant, i.e., 60 seconds, while the time interval of Shanghai is variable, as shown in Fig. 11. We also summarize the detailed statistics of both datasets in Table 1.

We divide each dataset into three parts with the splitting ratio of 7:1:2 as training set, validation set, and test set, respectively. Besides, we randomly retain $1 - ratio\%$ ($ratio = 30, 50, 70$) of the trajectory points for each trajectory. It means that $ratio\%$ of the trajectory points are missing for each trajectory. We call $ratio\%$ coverage ratio in the following of this paper.

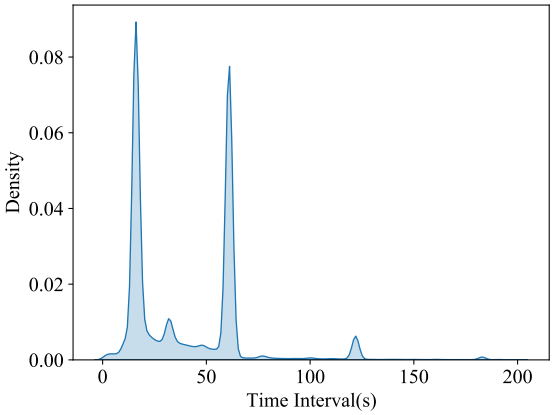


Fig. 11. Time Interval density of the Shanghai Taxi Dataset

Evaluation Methods. We evaluate the following methods.

– LI: Linear Interpolation, a common method for filling in the intermediate values of the independent variable. Here, we fill in the missing values of longitude and latitude independently. For the case that the beginning or the

Table 1. Statistics of Datasets After Preprocessing

Datasets	Porto	Shanghai
Duration	1 year	1 day
taxis	441	2218
trajectories	855167	64520
records	13293126	1853575
regions	23639	21795
time interval	1 minute	43.589 seconds (mean)
mean length	15.544	28.317

end of the trajectory are missing, we fill it as the nearest recorded trajectory point of the target incomplete trajectory.
- RST: The Raw Similar Trajectories generated by Rule-based Information Extractor, which is the result of integrating information from several other related incomplete trajectories and is simply a rigid combination of relevant spatial information.
- SAITS [6]: The state-of-the-art model in the field of time series imputation, which is based on the self-attention mechanism and trained by a joint-optimization approach.
- Transformer [22]: The model that is widely used in many fields and achieves remarkable performance. It is mainly composed of attention mechanism and feed forward network, and has an extremely powerful feature extraction ability. Here, we use the embeddings calculated by Eq. 4 as the input of Transformer.
- SimiDTR: Our proposed method.

Metrics. In the evaluation phase, we copy the recorded trajectory points to the corresponding position in $\boldsymbol{L}_{pre}$ (i.e., the predicted trajectory by model) to get $\boldsymbol{L}_{eva}$ (i.e., the finally recovered trajectory). Then we calculate the following metrics between $\boldsymbol{L}_{eva}$ and $\boldsymbol{L}_{tar}$ to evaluate the accuracy of the above methods. The following metrics are widely used to measure the similarity between trajectories in previous [15, 24, 25, 31].

- RMSE: The *Root Mean Squared Error* between two trajectory coordinates, which is used when the number of trajectory points is the same. In this paper, this metric is measured in kilometers.
- NDTW: The *Normalized Dynamic Time Warping*, where DTW [1] measures the similarity between trajectories of different lengths. NDTW is equal to DTW divided by the trajectory length.
- I-LCSS: The *Improved Longest Common SubSequence*, where LCSS [23] ignores distant points, so noise can be excluded. But it doesn't measure the distance between the near points, it's just a rough measure. Note that we subtract the original LCSS with 1 to get the I-LCSS used in this paper.
- EDR [3]: The *Edit Distance on Real sequence*, which excludes the influence of noise points and measures the near point on the basis of LCSS.

We set the space distance threshold of I-LCSS and EDR to 0.2 km. The smaller the RMSE, the NDTW, the LCSS, and the EDR are, the more accurate the method is.

Parameter Settings. In our experiment, we follow the reported optimal parameter settings by previous paper as running SAITS. In transformer and our model, the embedding dimmension d is set to 20, the number of encoder and decoder block B is set to 6 and the number of head of multi-head attention component H is set to 4. Hidden size of Feed Forward network d' is set to 256, and the dropout rate is set to 0.1. Before feeding numerical coordinates into SAITS, Transformer and SimiDTR, we firstly use min-max normalization to preprocess the numerical data.

5.2 Experimental Results

Overall. We report the performance of all the methods in Table. 2 and have the following observation:

- When the time interval is a fixed constant, LI and RST work better, even comparable to some deep learning models. However, when the time interval is not fixed, the performance of both of them is generally lower than that of the deep learning models.
- SAITS, as the representative of time series imputation methods, works well when the trajectory coverage ratio is small, but its performance decreases more significantly than the other deep learning models when the coverage ratio increases.
- Although Transformer only uses its own information when recovering incomplete trajectories, it has achieved pretty good results under all the three coverage ratios. From this point, we can see its strong characteristic learning ability.
- Due to the combination of similar trajectory information and the powerful feature extraction capabilities of deep neural networks when recovering incomplete trajectories, the performance of SimiDTR is significantly improved compared with all other methods above.

Ablation Analysis. We analyze the effects of all the three modules by removing one module at a time:

- **SimiDTR-S**: Only encoder and decoder are used to recover trajectories. In this case, we replace all the inputs of encoder and decoder with the embedding of incomplete trajectory. In other words, we only use the information contained in the incomplete trajectory itself to recover it.
- **SimiDTR-D**: Only rule-based information extractor and encoder are used to recover trajectories. In this case, we send the embedding of the tailored similar trajectory output by encoder into a linear layer to get its 2-dimensional trajectory coordinates. In other words, we regard the tailored similar trajectory as the trajectory of final recovered.

Table 2. Performance Comparison on two Datasets

Metric	Models	Porto			Shanghai		
		30%	50%	70%	30%	50%	70%
RMSE	LI	0.1486	0.2838	0.5306	0.3547	0.5221	0.7468
	RST	0.1391	0.2425	0.3761	0.4486	0.6647	0.9089
	SAITS	0.1385	0.2597	0.4749	0.2651	0.4575	0.7445
	Transformer	0.1290	0.1971	0.3260	0.2439	0.3168	0.4417
	SimiDTR	**0.0822**	**0.1343**	**0.1821**	**0.1750**	**0.2210**	**0.2769**
NDTW	LI	0.0649	0.1570	0.3349	0.1452	0.2663	0.4368
	RST	0.0525	0.1143	0.2013	0.1600	0.2761	0.4079
	SAITS	0.0606	0.1420	0.2911	0.0901	0.1989	0.3919
	Transformer	0.0410	0.0810	0.1513	0.0604	0.0970	0.1641
	SimiDTR	**0.0256**	**0.0536**	**0.0836**	**0.0395**	**0.0625**	**0.0906**
I-LCSS	LI	0.1173	0.2533	0.4232	0.0773	0.1734	0.3223
	RST	0.0916	0.1919	0.3040	0.1007	0.2082	0.3360
	SAITS	0.1117	0.2423	0.4048	0.0897	0.1880	0.3318
	Transformer	0.0787	0.1428	0.2301	0.0608	0.1072	0.1577
	SimiDTR	**0.0459**	**0.0983**	**0.1533**	**0.0514**	**0.0793**	**0.1132**
EDR	LI	0.1206	0.2662	0.4482	0.0804	0.1850	0.3469
	RST	0.0955	0.2071	0.3365	0.1061	0.2281	0.3820
	SAITS	0.1151	0.2566	0.4367	0.0925	0.2004	0.3628
	Transformer	0.0811	0.1518	0.2502	0.0638	0.1152	0.1717
	SimiDTR	**0.0478**	**0.1057**	**0.1697**	**0.0548**	**0.0873**	**0.1276**

The best result in each column is bold, while the second is underlined.

– **SimiDTR-E**: Only rule-based information extractor and decoder are used to recover trajectories. In this case, we directly send the raw similar trajectory obtained by the rule-based information extractor to the decoder without processing.

We report the results of these four models in Table. 3. We have the following obervation:

– It can be seen that SimiDTR with all modules has the best comprehensive performance, which also shows that each module can improve the recovery.
– Specifically, the three models with rule-based information extractor have a significant improvement compared with that without trajectory information when the coverage ratio is 70%, which shows the use of similar trajectory information can well alleviate the sparsity of trajectory data.
– In most cases, SimiDTR-D works better on I-LCSS and EDR than SimiDTR-E, which are the opposite of the effect on RMSE and NDTW. This phenomenon is even more pronounced on the Shanghai taxi dataset, where sampling intervals are not fixed.

Visualization Comparison. In order to analyze the effect of the model more vividly, we select two sample trajectories and visualize the recovery results of each model as shown in Fig. 12 and Fig. 13. Note that their coverage ratio are 70%, and they come from Shanghai taxi dataset and Porto dataset separately. We can observe that:

Table 3. Ablation Experiments on two Datasets

Metric	Models	Porto			Shanghai		
		30%	50%	70%	30%	50%	70%
RMSE	SimiDTR-S	0.1084	0.1895	0.3262	0.1811	0.2525	0.3643
	SimiDTR-D	0.1000	0.1658	0.2129	0.2210	0.3013	0.3503
	SimiDTR-E	0.0942	0.1488	0.1951	0.1813	0.2468	0.3107
	SimiDTR	**0.0822**	**0.1343**	**0.1821**	**0.1750**	**0.2210**	**0.2769**
NDTW	SimiDTR-S	0.0351	0.0775	0.1519	0.0408	0.0702	0.1189
	SimiDTR-D	0.0278	0.0590	0.0911	0.0501	0.0902	0.1278
	SimiDTR-E	0.0304	0.0613	0.0909	0.0427	0.0733	0.1080
	SimiDTR	**0.0256**	**0.0536**	**0.0836**	**0.0395**	**0.0625**	**0.0906**
I-LCSS	SimiDTR-S	0.0666	0.1367	0.2337	0.0612	0.0940	0.1342
	SimiDTR-D	0.0475	0.0987	0.1574	**0.0387**	0.0851	0.1232
	SimiDTR-E	0.0591	0.1182	0.1666	0.0565	0.0999	0.1312
	SimiDTR	**0.0459**	**0.0983**	**0.1533**	0.0514	**0.0873**	**0.1276**
EDR	SimiDTR-S	0.0687	0.1452	0.2541	0.0654	0.1032	0.1492
	SimiDTR-D	0.0496	0.1068	0.1743	**0.0412**	0.0950	0.1399
	SimiDTR-E	0.0615	0.1272	0.1838	0.0603	0.1098	0.1475
	SimiDTR	**0.0478**	**0.1057**	**0.1697**	0.0548	**0.0873**	**0.1276**

- In the case of sparsity, because an incomplete trajectory contains little information, Linear Interpolation, Transformer and SAITS are difficult to recover trajectory segments with serious data loss. However, with the help of similar trajectory information, our model can recover the movement trend of the object.
- The raw similar trajectory has a more similar change trend to the complete trajectory, but due to the spatial bias, temporal bias and temporal shifts between different trajectory data, it does not perform well in metrics. However, after it is sorted out by the neural network model, its effect will be greatly improved.

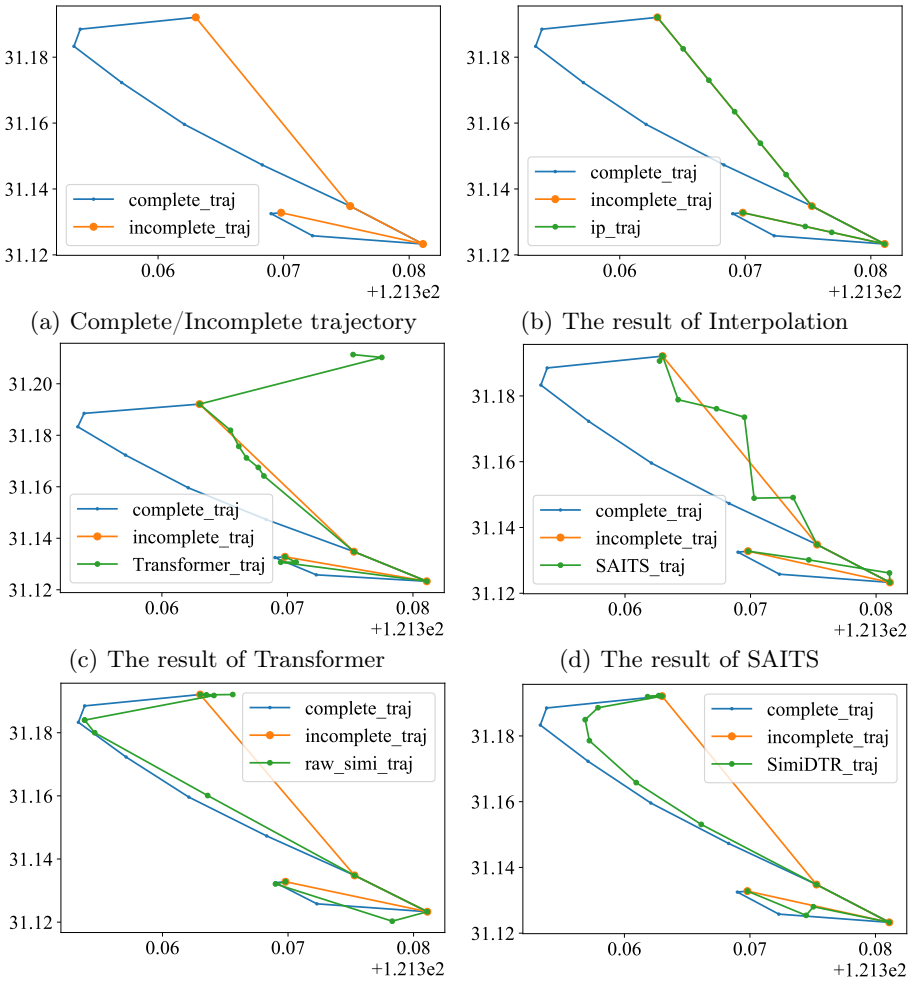
Efficiency. In order to analyze the efficiency of each part of the model, we tested the running time of the following parts separately:

- **Similarity**: The time of using rule-based information extractor to find the raw similar trajectory.
- **Processing**: The time of data processing, including min-max normalization, time feature mapping and trajectory alignment of each mini-batch.
- **Calculation**: The time of using trajectory embedding layer, encoder and decoder to get the final recovered trajectory.
- **Other**: The time spent in addition to the three parts of time above.
- **Total**: The time of recovering one incomplete trajectory.

The hardware environment of this experiment is as following: the operating system is Ubuntu 18.04.4 LTS, the processor is Intel(R) Xeon(R) CPU E5-2680

v4 2.40GHz, the memory size is 128G, the graphics card used is GeForce RTX 2080Ti, and the video memory size is 10G.

For each experiment, we randomly selected 512 trajectories in a dataset and calculate the average time of recovering one trajectory. When searching for raw similar trajectories, we use only one CPU kernel to search sequentially. The processing of data is also run by CPU. Recovering trajectories using deep models is run with GPU. A total of 10 such experiments are conducted, and we report their mean values on two datasets in Table 4. We have the following observations:



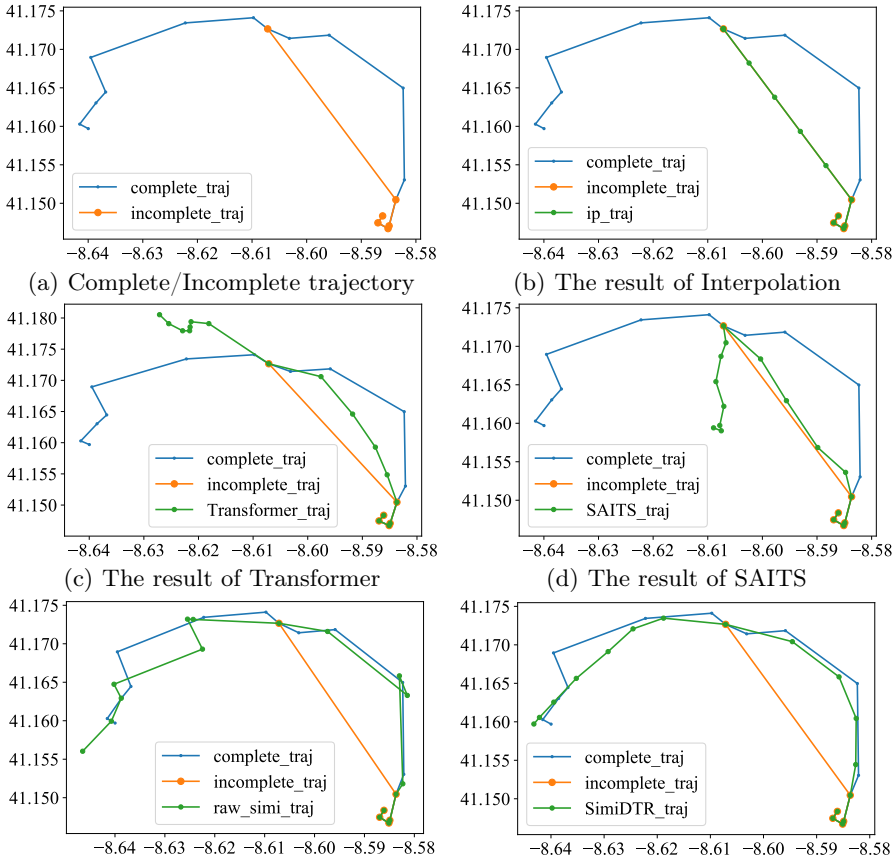
(a) Complete/Incomplete trajectory (b) The result of Interpolation

(c) The result of Transformer (d) The result of SAITS

(e) Raw similar trajectory (f) The result of SimiDTR

Fig. 12. A sample trajectory from Shanghai taxi dataset

– Time is spent mostly searching for raw similar trajectories, and it increases as the trajectory coverage ratio increases. This because the higher the coverage ratio, the more spatial information we need to search to fill in the missing trajectory points.
– Data processing and calculations with deep models in Porto dataset take less time than that in Shanghai taxi dataset. That is because the average length of the trajectories in Porto dataset is smaller than that in Shanghai taxi dataset.
– The number of trajectory points in Porto dataset is about 7.2 times that of the Shanghai taxi dataset, and the average time it takes to recover a trajectory is about 8.7 times that of Shanghai taxi dataset (the average time of the three coverage ratios).



(a) Complete/Incomplete trajectory (b) The result of Interpolation
(c) The result of Transformer (d) The result of SAITS
(e) Raw similar trajectory (f) The result of SimiDTR

Fig. 13. A sample trajectory from Porto dataset

Table 4. Runtime Analysis on two Datasets

Runtime	Porto			Shanghai		
	30%	50%	70%	30%	50%	70%
Similarity	72.811	121.206	151.346	6.227	10.285	15.976
Processing	1.982	1.941	1.921	2.213	2.183	2.211
Calculation	0.318	0.321	0.315	0.436	0.435	0.436
Other	0.026	0.029	0.027	0.027	0.026	0.028
Total	75.137	123.497	153.609	8.903	12.929	18.652

The above running time is measured by milliseconds.

6 Conclusion

We propose a novel deep trajectory recovery framework with enhanced trajectory similarity that enables the capture of complex mobility regularity for recovering incomplete trajectories.To counter data sparsity problems, the framework leverages the similarity between the target incomplete trajectory and other trajectories. In particular, it adopts a rule-based method that can extract a similar trajectory from the observed points of other incomplete trajectories for the given incomplete trajectory. Further, the framework exploits attention mechanism to generate a trajectory that is not actually existing but the most similar to the incomplete trajectory. The generated trajectory is used for recovering the incomplete trajectory. We conduct experiments on two datasets collected from real world. The experimental results show that this framework outperforms the state of the art in trajectory recovery performance, especially when data is sparse.

References

1. Agrawal, R., Faloutsos, C., Swami, A.: Efficient similarity search in sequence databases. In: FODO. pp. 69–84 (1993)
2. Cao, W., Wang, D., Li, J., Zhou, H., Li, L., Li, Y.: Brits: Bidirectional recurrent imputation for time series. NIPS **31** (2018)
3. Chen, L., Özsu, M.T., Oria, V.: Robust and fast similarity search for moving object trajectories. In: SIGMOD. pp. 491–502 (2005)
4. Cheng, C., Yang, H., Lyu, M.R., King, I.: Where you like to go next: Successive point-of-interest recommendation. In: IJCAI. pp. 2605–2611 (2013)
5. Cui, Y., Sun, H., Zhao, Y., Yin, H., Zheng, K.: Sequential-knowledge-aware next poi recommendation: A meta-learning approach. TOIS **40**(2), 1–22 (2021)
6. Du, W., Côté, D., Liu, Y.: Saits: Self-attention-based imputation for time series. arXiv preprint arXiv:2202.08516 (2022)
7. Fang, X., Huang, J., Wang, F., Zeng, L., Liang, H., Wang, H.: Constgat: Contextual spatial-temporal graph attention network for travel time estimation at baidu maps. In: SIGKDD. pp. 2697–2705 (2020)
8. Feng, J., Li, Y., Zhang, C., Sun, F., Meng, F., Guo, A., Jin, D.: Deepmove: Predicting human mobility with attentional recurrent networks. In: WWW. pp. 1459–1468 (2018)
9. Feng, S., Li, X., Zeng, Y., Cong, G., Chee, Y.M., Yuan, Q.: Personalized ranking metric embedding for next new poi recommendation. In: IJCAI. pp. 2062–2068 (2015)

10. Fortuin, V., Baranchuk, D., Rätsch, G., Mandt, S.: Gp-vae: Deep probabilistic time series imputation. In: AISTATS. pp. 1651–1661 (2020)
11. Gambs, S., Killijian, M.O., del Prado Cortez, M.N.: Next place prediction using mobility markov chains. In: MPM. pp. 1–6 (2012)
12. Guo, S., Chen, C., Wang, J., Liu, Y., Xu, K., Yu, Z., Zhang, D., Chiu, D.M.: Rod-revenue: Seeking strategies analysis and revenue prediction in ride-on-demand service using multi-source urban data. TMC **19**(9), 2202–2220 (2019)
13. Hadjieleftheriou, M., Kollios, G., Tsotras, V.J., Gunopulos, D.: Efficient indexing of spatiotemporal objects. In: EDBT. pp. 251–268 (2002)
14. Hung, C.C., Peng, W.C., Lee, W.C.: Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. PVLDB **24**(2), 169–192 (2015)
15. Liu, H., Wei, L.Y., Zheng, Y., Schneider, M., Peng, W.C.: Route discovery from mining uncertain trajectories. In: ICDMW. pp. 1239–1242 (2011)
16. Liu, Q., Wu, S., Wang, L., Tan, T.: Predicting the next location: A recurrent model with spatial and temporal contexts. In: AAAI. pp. 194–200 (2016)
17. Luo, Y., Cai, X., Zhang, Y., Xu, J., et al.: Multivariate time series imputation with generative adversarial networks. NIPS **31**, 1603–1614 (2018)
18. Luo, Y., Zhang, Y., Cai, X., Yuan, X.: E2gan: End-to-end generative adversarial network for multivariate time series imputation. In: IJCAI. pp. 3094–3100 (2019)
19. Ren, H., Ruan, S., Li, Y., Bao, J., Meng, C., Li, R., Zheng, Y.: Mtrajrec: Map-constrained trajectory recovery via seq2seq multi-task learning. In: SIGKDD. pp. 1410–1419 (2021)
20. Sun, H., Yang, C., Deng, L., Zhou, F., Huang, F., Zheng, K.: Periodicmove: Shift-aware human mobility recovery with graph neural network. In: CIKM. pp. 1734–1743 (2021)
21. Ulm, M., Brandie, N.: Robust online trajectory clustering without computing trajectory distances. In: ICPR. pp. 2270–2273 (2012)
22. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. NIPS **30**, 5998–6008 (2017)
23. Vlachos, M., Kollios, G., Gunopulos, D.: Discovering similar multidimensional trajectories. In: ICDE. pp. 673–684 (2002)
24. Wang, J., Wu, N., Lu, X., Zhao, W.X., Feng, K.: Deep trajectory recovery with fine-grained calibration using kalman filter. TKDE **33**(3), 921–934 (2019)
25. Wei, L.Y., Zheng, Y., Peng, W.C.: Constructing popular routes from uncertain trajectories. In: SIGKDD. pp. 195–203 (2012)
26. Xi, D., Zhuang, F., Liu, Y., Gu, J., Xiong, H., He, Q.: Modelling of bi-directional spatio-temporal dependence and users' dynamic preferences for missing poi check-in identification. In: AAAI. pp. 5458–5465. No. 01 (2019)
27. Xia, T., Qi, Y., Feng, J., Xu, F., Sun, F., Guo, D., Li, Y.: Attnmove: History enhanced trajectory recovery via attentional network. arXiv preprint arXiv:2101.00646 (2021)
28. Xie, M., Yin, H., Wang, H., Xu, F., Chen, W., Wang, S.: Learning graph-based poi embedding for location-based recommendation. In: CIKM. pp. 15–24 (2016)
29. Yoon, J., Zame, W.R., van der Schaar, M.: Estimating missing data in temporal data streams using multi-directional recurrent neural networks. TBME **66**(5), 1477–1490 (2018)
30. Yuan, J., Zheng, Y., Xie, X., Sun, G.: T-drive: Enhancing driving directions with taxi drivers' intelligence. TKDE **25**(1), 220–232 (2011)
31. Zheng, Y.: Trajectory data mining: an overview. TIST **6**(3), 1–41 (2015)